

Программирование Python

Лекция 3

Кафедра ИВТ и ПМ
ЗабГУ

2017

План

Прошлые темы

Циклы

Вложенные циклы

Списки

Операции

Генераторы списков

Сообщения об ошибках

Прошлые темы

- ▶ Зачем нужны циклические конструкции?
- ▶ Какие операторы цикла есть в python?

Прошлые темы

- ▶ Зачем нужны циклические конструкции?
- ▶ Какие операторы цикла есть в python?
- ▶ Как работает цикл **while**?

Прошлые темы

- ▶ Зачем нужны циклические конструкции?
- ▶ Какие операторы цикла есть в python?
- ▶ Как работает цикл **while**?

```
while условие:  
    оператор1  
else:  
    оператор2
```

Прошлые темы

- ▶ Как работает цикл **for**?

Прошлые темы

- ▶ Как работает цикл **for**?

```
for переменная in Последовательность:  
    оператор1  
else:  
    оператор2
```

Прошлые темы

- ▶ Как работает цикл **for**?

```
for переменная in Последовательность:  
    оператор1  
else:  
    оператор2
```

- ▶ Как работает функция `range(n)`?
- ▶ Как работает функция `range(a,b)`?
- ▶ Как создать последовательность из целых чисел от 0 до 100 с шагом 5?

Прошлые темы

- ▶ Как работает цикл **for**?

```
for переменная in Последовательность:  
    оператор1  
else:  
    оператор2
```

- ▶ Как работает функция `range(n)`?
- ▶ Как работает функция `range(a,b)`?
- ▶ Как создать последовательность из целых чисел от 0 до 100 с шагом 5?

```
range(0,101,5)
```

Прошлые темы

- ▶ Как напечатать все элементы списка?

```
l = [3.14, 512, 600, 2]
```

Прошлые темы

- ▶ Как напечатать все элементы списка?

```
l = [3.14, 512, 600, 2]
```

Способ 1

```
print(l)
```

```
[3.14, 512, 600, 2]
```

Прошлые темы

- ▶ Как напечатать все элементы списка?

```
l = [3.14, 512, 600, 2]
```

Способ 1

```
print(l)
```

```
[3.14, 512, 600, 2]
```

Способ 2

```
for e in l:  
    print(e, end='; ')
```

```
3.14; 512; 600; 2;
```

Прошлые темы

- ▶ Можно ли в теле цикла `for` менять переменную цикла?

Прошлые темы

- Можно ли в теле цикла `for` менять переменную цикла?

```
l = [10, 20, 30, 40]
for i in l:
    i = 0
    print(i)
```

Это допустимо.

Будет напечатано 4 нуля. При каждой новой итерации цикла значение `i` будет взято из последовательности.

Последовательность не изменится

Прошлые темы

- ▶ Как организовать цикл в теле которого можно изменять последовательность?

Прошлые темы

- ▶ Как организовать цикл в теле которого можно изменять последовательность?

```
for i in range( len(l) ):
    l[i] = random()
```


Прошлые темы

- ▶ Как организовать цикл в теле которого можно изменять последовательность?

```
for i in range( len(l) ):
    l[i] = random()
```

- ▶ Сколько раз выполнится этот цикл?

```
sum = 0
i = 1
e = 1
while e>0.1:
    e = 1/i
    sum = sum + e
```

Прошлые темы

- ▶ Как организовать цикл в теле которого можно изменять последовательность?

```
for i in range( len(l) ):
    l[i] = random()
```

- ▶ Сколько раз выполнится этот цикл?

```
sum = 0
i = 1
e = 1
while e>0.1:
    e = 1/i
    sum = sum + e
```

Этот цикл будет выполняться бесконечно

Прошлые темы

- ▶ Сколько раз выполнится этот цикл?

```
sum = 0
i = 1
e = 1
while e>0.1:
    e = 1/i
    sum = sum + e
    print(i, ": ", e)
    i = i + 1
```

Прошлые темы

- ▶ Сколько раз выполнится этот цикл?

```
sum = 0
i = 1
e = 1
while e>0.1:
    e = 1/i
    sum = sum + e
    print(i, ": ", e)
    i = i + 1
```

10 раз

- ▶ Какие существуют операторы управления циклами?
- ▶ Для чего они нужны?

Прошлые темы

- ▶ Как работает оператор **continue**?
- ▶ Как работает оператор **break**?

Прошлые темы

- ▶ Как работает оператор **continue**?
- ▶ Как работает оператор **break**?

```
e = 100
while e>1:
    e = e / 2
    if random() > 0.5:
        break
    print(e)
```

- ▶ Как организовать этот цикл без использования оператора **break**?

Прошлые темы

- ▶ Как организовать этот цикл без использования оператора `break`?

Прошлые темы

- ▶ Как организовать этот цикл без использования оператора break?

```
e = 100
flag = True
while e>1 and flag:
    e = e / 2
    if random() <= 0.5:  # условие изменилось
        print(e)        # на противоположное
    else:
        flag = False
```


Флаг - переменная (обычно логическая) которая характеризует состояние некоторого объекта или процесса.

Прошлые темы

- ▶ Что такое список?
- ▶ Как задать пустой список?

Прошлые темы

- ▶ Что такое список?
- ▶ Как задать пустой список?
`l = []`
- ▶ Как задать список из заранее известных элементов?

Прошлые темы

- ▶ Что такое список?
- ▶ Как задать пустой список?
`l = []`
- ▶ Как задать список из заранее известных элементов?
`l = [20, 8, -50, 6.125]`
- ▶ Как добавить элемент в конец списка?

Прошлые темы

- ▶ Что такое список?
- ▶ Как задать пустой список?
- ▶ Как задать список из заранее известных элементов?
- ▶ Как добавить элемент в конец списка?

```
l = []
```

```
l = [20, 8, -50, 6.125 ]
```

```
l = l + [ 16 ]
```

здесь к списку добавляется список из одного элемента

или

```
l.append( 16 )
```

Прошлые темы

- ▶ Как создать список из последовательности?

Прошлые темы

- ▶ Как создать список из последовательности?

```
l = list( range(0, 22, 3) )
```

Прошлые темы

`l = [20, 8, -50, 6.125]`

- ▶ Как обратиться к n-му элементу списка?

Прошлые темы

`l = [20, 8, -50, 6.125]`

- ▶ Как обратиться к n-му элементу списка?

`l [n-1]`

`l [3] # 6.125`

- ▶ Как обратиться к предпоследнему элементу списка?

Прошлые темы

```
l = [20, 8, -50, 6.125 ]
```

- ▶ Как обратиться к n-му элементу списка?

```
l [ n-1 ]
```

```
l [ 3 ] # 6.125
```

- ▶ Как обратиться к предпоследнему элементу списка?

```
l [ -2 ] # -50
```

Прошлые темы

```
l = [20, 8, -50, 6.125 ]
```

- ▶ Как обратиться к n-му элементу списка?

```
l [ n-1 ]
```

```
l [ 3 ] # 6.125
```

- ▶ Как обратиться к предпоследнему элементу списка?

```
l [ -2 ] # -50
```

- ▶ Что такое срез?

Прошлые темы

`l = [20, 8, -50, 6.125]`

- ▶ Как обратиться к n-му элементу списка?

`l [n-1]`

`l [3] # 6.125`

- ▶ Как обратиться к предпоследнему элементу списка?

`l [-2] # -50`

- ▶ Что такое срез?
- ▶ Как получить элементы списка с первого (по порядку) до третьего?

Прошлые темы

```
l = [20, 8, -50, 6.125 ]
```

- ▶ Как обратиться к n-му элементу списка?

```
l [ n-1 ]
```

```
l [ 3 ] # 6.125
```

- ▶ Как обратиться к предпоследнему элементу списка?

```
l [ -2 ] # -50
```

- ▶ Что такое срез?

- ▶ Как получить элементы списка с первого (по порядку) до третьего?

```
l [ 0 : 4 ]
```

Прошлые темы

- ▶ Что будет если обратиться к несуществующему элементу списка?

```
l[1000000]
```

Прошлые темы

- ▶ Что будет если обратиться к несуществующему элементу списка?

```
l[1000000]
```

Ошибка индексации. Программа аварийно завершит свою работу.

`IndexError: list index out of range`

- ▶ Что будет если в срезе обратиться к номерам несуществующих элементов?

```
l[ 1000:2000 ]
```

Прошлые темы

- ▶ Что будет если обратиться к несуществующему элементу списка?

```
l[1000000]
```

Ошибка индексации. Программа аварийно завершит свою работу.

```
IndexError: list index out of range
```

- ▶ Что будет если в срезе обратиться к номерам несуществующих элементов?

```
l[ 1000:2000 ]
```

Результатом этой операции будет пустой список

```
[]
```


Outline

Прошлые темы

Циклы

Вложенные циклы

Списки

Операции

Генераторы списков

Сообщения об ошибках

Вложенные циклы

Циклы позволяют повторять выполнение любого набора операторов.

В частности можно повторять много раз выполнение другого цикла.

Такие циклы называются вложенными.

```
заголовок_цикла1 :  
    заголовок_цикла2 :  
        операторы
```

Цикл 1 называют **внешним**

Цикл 2 называют **вложенным** или **внутренним** и принадлежит телу цикла 1.

Вложенные циклы

Напечатать числа в виде следующей таблицы

1 2 3

4 5 6

7 8 9

```
n = 0
for i in range(1,4):      # отвечает за строки таблицы
    for j in range(1,4):  # отвечает за элементы в строке
        print( n, end=" ")
        n = n + 1
    # переход на новую строку
    # в конце каждой итерации внешнего цикла
    print()
```

Вложенные циклы

```
for i in range(1,4):      # отвечает за строки таблицы
    for j in range(1,4):  # отвечает за элементы в строке
        print( i, j, end=" ")
    # переход на новую строку
    # в конце каждой итерации внешнего цикла
    print()
```

Что будет напечатано в результате?

Вложенные циклы

```
for i in range(1,4):      # отвечает за строки таблицы
    for j in range(1,4):  # отвечает за элементы в строке
        print( i, j, end=" ")
    # переход на новую строку
    # в конце каждой итерации внешнего цикла
    print()
```

Что будет напечатано в результате?

```
11 12 13
21 22 23
31 32 33
```

Вложенные циклы

Можно сделать переменную вложенного цикла зависимой от внешнего цикла

```
for i in range(1,5):  
    print(i,": ", end="")  
    for j in range(1,i+1):  
        print( j, end="")  
    print()
```

Что будет напечатано в результате?

Вложенные циклы

Можно сделать переменную вложенного цикла зависимой от внешнего цикла

```
for i in range(1,5):  
    print(i,": ", end="")  
    for j in range(1,i+1):  
        print( j, end="")  
    print()
```

Что будет напечатано в результате?

```
1: 1  
2: 12  
3: 123  
4: 1234
```

Вложенные циклы

Вложенные циклы удобно использовать для работы с вложенными списками

Найдём сумму элементов матрицы

```
M = [ [1, 2, 3],  
       [4, 5, 6],  
       [7, 8, 9]]
```

```
s = 0  
for line in M:           # отвечает за строки  
    for e in line:       # отвечает за элементы в строке  
        s = s + e  
  
print(s)    # 45
```


Вложенные циклы

Какую работу выполняет следующий сниппет?

```
M = [ [1, 2, 3],  
      [3, 2, 2],  
      [3, 2, 3]]
```

```
a = []  
for line in M:                # отвечает за строки  
    mx = 1  
    for e in line:            # отвечает за элементы в строке  
        mx = mx * e  
    a = a + [mx]  
  
print(a)
```

Вложенные циклы

Какую работу выполняет следующий сниппет?

```
M = [ [1, 2, 3],  
      [3, 2, 2],  
      [3, 2, 3]]
```

```
a = []  
for line in M:                # отвечает за строки  
    mx = 1  
    for e in line:            # отвечает за элементы в строке  
        mx = mx * e  
    a = a + [mx]  
  
print(a)
```

Что будет напечатано?

Вложенные циклы

Какую работу выполняет следующий снипплет?

```
M = [ [1, 2, 3],  
      [3, 2, 2],  
      [3, 2, 3]]
```

```
a = []  
for line in M:                # отвечает за строки  
    mx = 1  
    for e in line:            # отвечает за элементы в строке  
        mx = mx * e  
    a = a + [mx]  
  
print(a)
```

Что будет напечатано?

```
[6, 12, 18]
```

Вложенные циклы

Для создания вложенных циклов можно использовать while
Какую работу выполняет следующий сниппет?

```
M = []
stop = False
while not stop:
    line = []
    while True:
        e = input()
        if e == ';':
            break
        elif e == '.':
            stop = True
            break
    line = line + [ float(e) ]
M = M + [line]
```

Вложенные циклы

- ▶ Для организации вложенных циклов можно использовать операторы `while`, `for` или их произвольную комбинацию
- ▶ Можно организовывать сколь угодно много уровней вложенности

При этом общее число итераций равно произведению числа итераций всех вложенных циклов.

Поэтому вложенные циклы с ресурсоёмкими операциями внутри могут выполняться значительное время.

- ▶ Операторы **`break`** и **`continue`** управляют выполнением только того цикла, в теле которого вызваны.
- ▶ Вложенные циклы хорошо подходят для обработки вложенных списков
- ▶ Однако *создание* списков лучше поручить генераторам списков (см. далее)

Outline

Прошлые темы

Циклы

Вложенные циклы

Списки

Операции

Генераторы списков

Сообщения об ошибках

Операции со списками

```
help( list )
```

```
l = [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

► Удаление элементов

► Удалить один элемент

```
del l[2]
```

► Удалить несколько элементов

```
del l[1:3]
```

```
del l[1:10:2]
```

Будет удалён каждый второй элемент начиная со второго
(по порядку)

```
[1, 3, 5, 7, 9]
```

► Удалить список целиком

```
del l
```

Удаление

Оператор `del` способен удалять не только списки, но и любые другие переменные.

Память занимаемая переменной освобождается.

Операции со списками

```
l = [10, 20, 30, 40, 50]
```

- ▶ Удаление элементов

`l.pop()` - удаляет последний элемент в списке и возвращает его значение

`l.pop(индекс)` - удаляет элемент в списке по индексу и возвращает его значение

Операции со списками

```
l = [1, 2, 3, 4, 5]
```

- ▶ Добавление элементов в конец списка

```
l.extend ( список )  
l.extend ( [ 22, 33, 44 ] )  
# параметром метода extend должен быть список  
  
# l = [1, 2, 3, 4, 5, 22, 33, 44 ]  
  
# та же самая операция  
l = l + [ 22, 33, 44 ]
```

Операции со списками

```
l = [1, 2, 3, 4, 5]
```

- ▶ Вставка элементов в произвольную позицию

```
l.insert( позиция, элемент )
```

```
l.insert( 2, 987 )
```

```
# l = [1, 2, 987, 3, 4, 5]
```

При вставке элемент стоящий на данной позиции сдвигается вправо освобождая место для нового.

Операции со списками

`l = [1, 2, 3, 4, 5]`

- ▶ Вставка элементов в произвольную позицию
Вставка элементов используя срезы

`l[2:2] = [987]`

l = [1, 2, 987, 3, 4, 5]

Срез должен быть пустым списком.

Справа от оператора присваивания должен быть список.

Операции со списками

`l = [1, 2, 3, 4, 5]`

- ▶ Вставка элементов в произвольную позицию (с заменой)
Вставка элементов используя срезы

`# l[2:3] = [987, 456]`

`# l = [1, 2, 987, 456, 4, 5]`

Срез - элементы которые будут заменены на новые.

Справа от оператора присваивания должен быть список.

Число заменяемых элементов может быть не равно числу новых.

Операции со списками

- Поиск индекса элемента

`l.index(значение)` - возвращает смещение первого элемента (который равен значению) в списке.

```
i = l.index( 40 )  
# i = 3
```

`l.index(значение, start, stop)` - возвращает смещение первого элемента (который равен значению) в списке. Поиск производится от позиции `start` до позиции `stop`, включая последнюю.

Операции со списками

```
l = [1, 2, 3, 4, 5]
```

```
l.index( 78956 )
```

Если элемент не найден - происходит ошибка времени выполнения:

```
ValueError: 78956 is not in list
```

см. обработку исключительных ситуаций в Python

Операции со списками

```
l = [1, 2, 3, 4, 3, 5, 5, 5]
```

- Поиск числа вхождений `l.count(значение)` - возвращает число вхождений *значения* в список.

```
l.count( -8 )    # -> 0
```

```
l.count( 2 )     # -> 1
```

```
l.count( 5 )     # -> 3
```


Операции со списками

- ▶ Сортировка `l.sort()` - сортирует список по возрастанию.

```
l = [4, 2, -2, 2, 2, 5]  
l.sort()
```

```
# [-2, 2, 2, 2, 4, 5]
```

`l.sort(reverse = True)` Сортировка в обратном направлении. Для элементов списка должна быть

определена операция сравнения

```
l3 = [10.3 , 'd', 'acb']  
l3.sort()
```

`TypeError: unorderable types: str() < float()`

Ошибка: нельзя сравнить значения типа строка и вещественное число

Операции со списками

- Изменение порядка элементов на обратный `l.reverse()`

```
l = [-2, 2, 2, 2, 4, 5]
```

```
l.reverse()
```

```
# [5, 4, 2, 2, 2, -2]
```

Операции со списками

Изменить порядка элементов на обратный можно сделав срез
Однако в этом случае создаётся *новый* инвертированный
список, в дополнение к неизменному старому.

```
l = [-2, 2, 2, 2, 4, 5]
```

```
l2 = l[::-1]
```

Операции со списками

Создание поверхностной копии (shallow copy) списка

```
1 = [-2, 2, 2, 2, 4, 5]
```

```
12 = 1.copy()
```

```
# аналогичное действие с помощью среза
```

```
12 = 1[:]
```

shallow copy vs deep copy vs assignment

Операции со списками

Операции со списками можно применять не только к переменным типа `list` но и к литералам

Операции со списками

► Конкатенация

```
l = [1,2,3,4]
```

```
l2 = [20, 40]
```

```
l3 = l + l2 # [1,2,3,4, 20, 40]
```

► Повторение элементов

```
l = ['ha ']*2 # ['ha ', 'ha ']
```

```
l2 = l * 2 + [-7, 5]*2 # ['ha ', 'ha ', 'ha ', 'ha ', -7, 5, -7, 5]
```

Операции со списками

- Проверка на вхождение

```
l = [1,2,3,4]
```

```
4 in l      # True
```

```
79 in l     # False
```

```
"b" in l    # False
```

```
a = 3
```

```
a in l      # True
```

Операции со списками

- ▶ Проверка на вхождение Оператор **in** часто используют в условном операторе:

```
ans = input("Для продолжения введите Да или Yes")
```

```
ans = ans.lower()    # преобразуем строку в нижний регистр
if ans in ["да", "д", "yes", "y"] :
    print ( "Продолжаем...")
```


Outline

Прошлые темы

Циклы

Вложенные циклы

Списки

Операции

Генераторы списков

Сообщения об ошибках

Динамическое создание списка

Чаще всего списки необходимо создавать автоматически.

Во-первых число элементов в списке может быть неизвестным и определяться в процессе наполнения списка.

Во-вторых между элементами списка и или их порядковым номером может быть некая зависимость, которую логично именно запрограммировать, а не описывать вручную каждый элемент.

Лучший способ создать список - использовать не цикл for или while, а генератор списков.

Генератор списков (list comprehension) - специальная конструкция языка для *создания* списков.

Генераторы списков

Синтаксис:

[выражение for перемен. in последовательность предикат]

Для каждого значения *переменной* из *последовательности* будет вычислено *выражение* и добавлено в список если предикат истинен.

Предикат можно опустить

Предикат (прогр.) - выражение, результат которого истина или ложь. За счёт внутренних оптимизаций генератор списков - это самый быстрый способ динамически создать список.

Генераторы списков

Создать список из квадратов чисел от 2 до 7

```
l = [ x**2 for x in range(2,8) ]
```

```
# l = [4, 9, 16, 25, 36, 49]
```

Для каждого значения x в последовательности `range(2,8)` будет вычислено выражение $x**2$, которое будет добавлено в список.

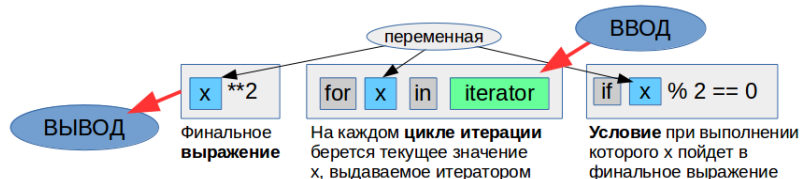
Синтаксически генераторы списков напоминают циклические конструкции, но тело цикла здесь находится перед заголовком цикла и всегда состоит из одного выражения

Генераторы списков

Создать список из квадратов только *чётных* чисел от 2 до 7

```
l = [x**2 for x in range(2,8) if x % 2 == 0]
```

```
# l = [4, 16, 36]
```



Генераторы списков

Для лучшей читаемости генератор списка стоит расположить на нескольких строках

```
l = [ x**2  
      for x in range(2,8)  
      if x % 2 == 0 ]
```

```
# l = [4, 16, 36]
```

Отступы в этом случае не создают вложенного блока.

Генераторы списков

Вложенные циклы

```
[ выражение  
  for Y in последовательность2  
  for X in последовательность1  
  предикат]
```

X, Z - переменные цикла

оператор `for X in последовательность1` является вложенным по отношению к `for Y in последовательность2`

Таким образом вложенные циклы в генераторах списков организуются таким же образом, что и обычные вложенные циклы

Генераторы списков

Задача: получить из вложенных списков (матрицы) - линейный список

```
matrix = [[0, 1, 2, 3],  
          [10, 11, 12, 13],  
          [20, 21, 22, 23]]
```

```
flattened = [n for row in matrix for n in row]  
# row - строка матрицы
```

```
print(flattened)  
# [0, 1, 2, 3, 10, 11, 12, 13, 20, 21, 22, 23]
```


Генераторы списков

Задача: создать матрицу

```
M = [ [ i*10+j for j in range(3) ] for i in range(3) ]
```

Генераторы списков

Задача: создать матрицу

```
M = [ [ i*10+j for j in range(3) ] for i in range(3) ]
```

Как будет выглядеть матрица?

```
[[0, 1, 2],  
 [10, 11, 12],  
 [20, 21, 22]]
```

Сообщения о синтаксических ошибках

На какие две категории можно разделить ошибки в программах?

Сообщения о синтаксических ошибках

На какие две категории можно разделить ошибки в программах?

- ▶ Ошибки выявляемые компилятором или интерпретатором
- ▶ Логические ошибки
Эти ошибки должен выявлять программист

NameError

Что общего у всех этих операторов?

```
print( x )  
b = a * 2  
c = sin( b )  
z = randint(0,99)  
sleep(10)  #приостанавливает программу на 10 секунд
```

Предполагается, что в этом сниппете никакие другие операторы не пропущены.

NameError

Что общего у всех этих операторов?

```
print( x )  
b = a * 2  
c = sin( b )  
z = randint(0,99)  
sleep(10)  #приостанавливает программу на 10 секунд
```

Предполагается, что в этом сниппете никакие другие операторы не пропущены.

NameError: name 'x' is not defined.

NameError: name 'a' is not defined.

NameError: name 'sin' is not defined.

NameError: name 'randint' is not defined.

NameError: name 'sleep' is not defined.

NameError - обращение к не объявленным именам
(идентификатор)

SyntaxError - общие ошибки синтаксиса

```
name = "Петя"  
print name
```

SyntaxError - общие ошибки синтаксиса

```
name = "Петя"  
print name
```

SyntaxError: invalid syntax

Параметры функции должны быть в круглых скобках.

Правильно:

```
print( name )
```

Примечание: синтаксис второй версии Python допускал такой способ вывода переменной на экран.

SyntaxError - общие ошибки синтаксиса

```
from random import randint
a = randint(0,9)
if a == 5
    print("а равно пяти")
```

SyntaxError: invalid syntax

SyntaxError - общие ошибки синтаксиса

```
from random import randint
a = randint(0,9)
if a == 5
    print("a равно пяти")
```

SyntaxError: invalid syntax

Перед каждым блоком в составном операторе должно стоять двоеточие

Правильно:

```
if a == 5 :
    print("a равно пяти")
```

SyntaxError - общие ошибки синтаксиса

```
from random import randint  
a = randint(0,9)  
if a = 5 :  
    print("а равно пяти")
```

SyntaxError - общие ошибки синтаксиса

```
from random import randint
a = randint(0,9)
if a = 5 :
    print("a равно пяти")
```

SyntaxError: invalid syntax

В условном операторе должно быть логическое выражение.
Здесь - вместо оператора проверки на равенство использован оператор сравнения.

Правильно:

```
if a == 5 :
    print("a равно пяти")
```

IndentationError - ошибки связанные с отступами

```
a = 10  
if a > 0:  
print(a)
```

IndentationError - ошибки связанные с отступами

```
a = 10
if a > 0:
print(a)
```

IndentationError: expected an indented block.

```
a = 10
if a > 0:
    print(a)
```

TabError - ошибки связанные с отступами

```
a = 10  
    print(a)
```

TabError: inconsistent use of tabs and spaces in indentation

Правильно

```
a = 10  
print(a)
```

TypeError - ошибки типа

```
a = input("a> ")  
b = a ** 3
```


TypeError - ошибки типа

```
a = input("a> ")  
b = a ** 3
```

TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'int' Правильно:

```
a = float( input("a> ") )  
b = a ** 3
```

Ссылки и литература

- ▶ Всё о выражениях-генераторах, генераторах списков, множествах и словарей

Ссылки и литература

ЭБС

- ▶ biblio-online.ru - ЭБС Юрайт
- ▶ studentlibrary.ru - ЭБС "КОНСУЛЬТАНТ СТУДЕНТА"
- ▶ Федоров, Д. Ю. Программирование на языке высокого уровня python : учебное пособие для прикладного бакалавриата Содержит краткое описание языка.
- ▶ ru.wikibooks.org/wiki/Python - Викиучебник
- ▶ Лутц М. Изучаем Python. 2010. - 1280 с. Содержит подробное описание языка.
- ▶ Официальная документация Python3
`help(имя)`

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming